

Boosting Methods

Jeremy Cohen

Princeton University

3 May 2018

Overview

- 1 AdaBoost
 - Forward Stagewise Additive Modeling
 - The Loss Function
- 2 Selecting a Loss Function
 - Classification
 - Regression
- 3 Boosting Trees
 - Brief Background on CART
 - Boosting Trees
- 4 Gradient Boosting
 - Steepest Descent
 - Gradient Boosting
- 5 Tuning and Metaparameter Values
 - Tree Size
 - Regularization
- 6 Implementation in R

- 1 AdaBoost
 - Forward Stagewise Additive Modeling
 - The Loss Function
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Original boosting algorithm designed for the binary classification problem.

- Given an output variable, $Y \in \{-1, 1\}$ and a vector of predictor variables, X , a classifier $G(X)$ produces a prediction taking one of the two values of Y

¹A weak classifier is one that performs only slightly better than random guessing.

Original boosting algorithm designed for the binary classification problem.

- Given an output variable, $Y \in \{-1, 1\}$ and a vector of predictor variables, X , a classifier $G(X)$ produces a prediction taking one of the two values of Y
- We begin with a "weak" classifier fit to the data.¹

¹A weak classifier is one that performs only slightly better than random guessing.

Original boosting algorithm designed for the binary classification problem.

- Given an output variable, $Y \in \{-1, 1\}$ and a vector of predictor variables, X , a classifier $G(X)$ produces a prediction taking one of the two values of Y
- We begin with a "weak" classifier fit to the data.¹
- We then reweight the observations so that those that were misclassified in this round, $y_i \neq G(x_i)$ are upweighted

¹A weak classifier is one that performs only slightly better than random guessing.

Original boosting algorithm designed for the binary classification problem.

- Given an output variable, $Y \in \{-1, 1\}$ and a vector of predictor variables, X , a classifier $G(X)$ produces a prediction taking one of the two values of Y
- We begin with a "weak" classifier fit to the data.¹
- We then reweight the observations so that those that were misclassified in this round, $y_i \neq G(x_i)$ *are upweighted*
- We then fit a new classifier and repeat this M times.

¹A weak classifier is one that performs only slightly better than random guessing.

Original boosting algorithm designed for the binary classification problem.

- Given an output variable, $Y \in \{-1, 1\}$ and a vector of predictor variables, X , a classifier $G(X)$ produces a prediction taking one of the two values of Y
- We begin with a "weak" classifier fit to the data.¹
- We then reweight the observations so that those that were misclassified in this round, $y_i \neq G(x_i)$ *are upweighted*
- We then fit a new classifier and repeat this M times.
- The final classification is given by a weighted vote of all classifiers, with those $G_m(x)$ that are more accurate receiving higher weights.

¹A weak classifier is one that performs only slightly better than random guessing.

AdaBoost, Visually

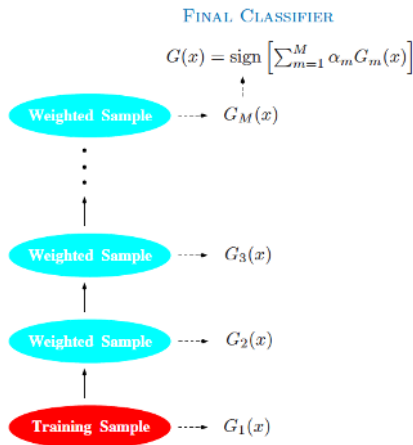


Figure 1: (Hastie et al. 2009:338)

AdaBoost, More Formally

- 1 Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \dots, N$

AdaBoost, More Formally

- 1 Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \dots, N$
- 2 For $m = 1$ to M :

AdaBoost, More Formally

- ① Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \dots, N$
- ② For $m = 1$ to M :
 - Fit a classifier $G_m(x)$ to the training data using weights w_i

AdaBoost, More Formally

- ① Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \dots, N$
- ② For $m = 1$ to M :
 - Fit a classifier $G_m(x)$ to the training data using weights w_i
 - Compute the error of the classifier as

$$err_m = \frac{\sum_{i=1}^N w_i \mathbf{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$$

AdaBoost, More Formally

- ① Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \dots, N$
- ② For $m = 1$ to M :
 - Fit a classifier $G_m(x)$ to the training data using weights w_i
 - Compute the error of the classifier as

$$err_m = \frac{\sum_{i=1}^N w_i \mathbf{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$$

- Compute $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$

AdaBoost, More Formally

- ① Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \dots, N$
- ② For $m = 1$ to M :
 - Fit a classifier $G_m(x)$ to the training data using weights w_i
 - Compute the error of the classifier as

$$err_m = \frac{\sum_{i=1}^N w_i \mathbf{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$$

- Compute $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$
- Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot G_m(x_i)]$

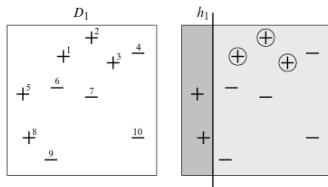
AdaBoost, More Formally

- ① Initialize the observation weights $w_i = \frac{1}{N}$, for $i = 1, 2, \dots, N$
- ② For $m = 1$ to M :
 - Fit a classifier $G_m(x)$ to the training data using weights w_i
 - Compute the error of the classifier as

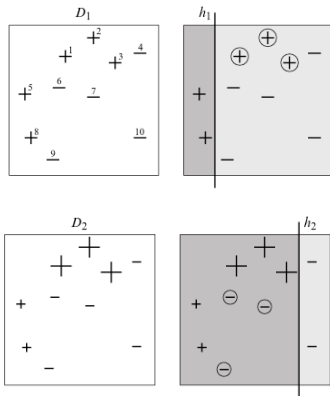
$$err_m = \frac{\sum_{i=1}^N w_i \mathbf{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$$

- Compute $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$
 - Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot G_m(x_i)]$
- ③ Output $G(x) = \text{sign} \left[\sum_{m=1}^M \alpha_m G_m(x) \right]$

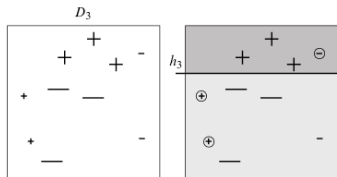
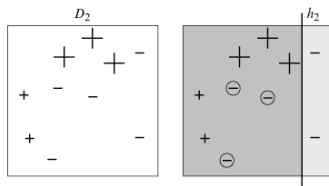
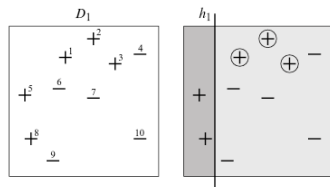
AdaBoost, Visually in More Detail



AdaBoost, Visually in More Detail



AdaBoost, Visually in More Detail



AdaBoost, Visually in More Detail

$$H = \text{sign} \left(0.42 \begin{array}{|c|} \hline \text{[Diagram: Left column shaded, right column white]} \\ \hline \end{array} + 0.65 \begin{array}{|c|} \hline \text{[Diagram: Left column white, right column shaded]} \\ \hline \end{array} + 0.92 \begin{array}{|c|} \hline \text{[Diagram: Top row shaded, bottom row white]} \\ \hline \end{array} \right)$$

$$= \begin{array}{|c|c|c|c|} \hline \text{[Diagram: 4x4 grid with shaded cells and +/- signs]} \\ \hline \end{array}$$

- 1 AdaBoost
 - Forward Stagewise Additive Modeling
 - The Loss Function
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Boosting Implements an Additive Model

- What is AdaBoost doing and how can we generalize it to other classification problems and to regression?

Boosting Implements an Additive Model

- What is AdaBoost doing and how can we generalize it to other classification problems and to regression?
- It turns out that AdaBoost implements Forward Stagewise Additive Modeling (FSAM) using an exponential loss function. Let's take these one at a time.

Forward Stagewise Additive Modeling

- 1 Initialize $f_0(x) = 0$

Forward Stagewise Additive Modeling

- 1 Initialize $f_0(x) = 0$
- 2 For $m = 1$ to M :

Forward Stagewise Additive Modeling

- ① Initialize $f_0(x) = 0$
- ② For $m = 1$ to M :
 - Compute:

$$(\beta_m, \gamma_m) = \underset{\beta, \gamma}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i); \gamma)$$

Forward Stagewise Additive Modeling

- ① Initialize $f_0(x) = 0$
- ② For $m = 1$ to M :
 - Compute:

$$(\beta_m, \gamma_m) = \underset{\beta, \gamma}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$

- Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

Forward Stagewise Additive Modeling

- 1 Initialize $f_0(x) = 0$
- 2 For $m = 1$ to M :
 - Compute:

$$(\beta_m, \gamma_m) = \operatorname{argmin}_{\beta, \gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$

- Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

At each iteration, we fit the optimal basis function and corresponding coefficient β_m to add to the current expansion, $f_{m-1}(x)$. We do not update the parameters of previously estimated functions

- 1 AdaBoost
 - Forward Stagewise Additive Modeling
 - The Loss Function
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

The Loss Function

- AdaBoost minimizes an exponential loss function. But let's look at the more familiar case of squared-error loss familiar from linear regression

$$\begin{aligned}L(y, f(x)) &= (y - f(x))^2 \\L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\&= (r_{im} - \beta b(x_i; \gamma))^2\end{aligned}$$

The Loss Function

- AdaBoost minimizes an exponential loss function. But let's look at the more familiar case of squared-error loss familiar from linear regression

$$\begin{aligned}L(y, f(x)) &= (y - f(x))^2 \\L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\&= (r_{im} - \beta b(x_i; \gamma))^2\end{aligned}$$

- So the basis function that best fits the residuals from the *last* iteration will minimize the loss and be added to the model.

- 1 AdaBoost
- 2 Selecting a Loss Function
 - Classification
 - Regression
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Selecting a Loss Function

- Will depend on the problem at hand.

- 1 AdaBoost
- 2 Selecting a Loss Function
 - Classification
 - Regression
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Loss Functions for Classification

- In the binary classification situation outlined above, the margin, $yf(x)$, plays a role analogous to the residuals $y - f(x)$ in regression. Positive margins are classified correctly; negative margins are misclassified.
- We want a loss function that penalizes negative margins more heavily than positive ones.

Loss Functions for Classification

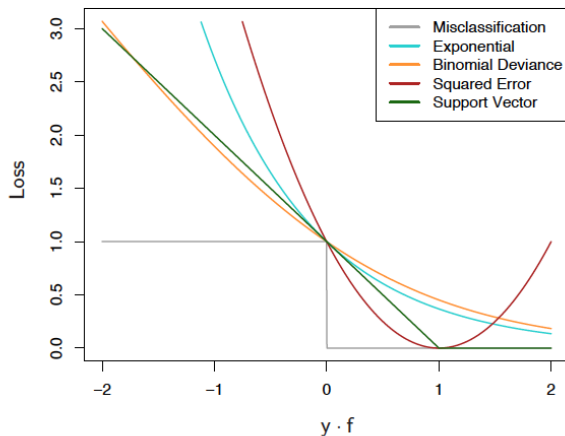


Figure 2: (Hastie et al. 2009:347)

- 1 AdaBoost
- 2 Selecting a Loss Function
 - Classification
 - Regression
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Loss Functions for Regression

- In the regression setting we can look at squared error loss (seen above) or absolute loss, $L(y, f(x)) = |y - f(x)|$
- We can also use the Huber loss criterion used in robust regression to address the strengths of each

Loss Functions for Regression

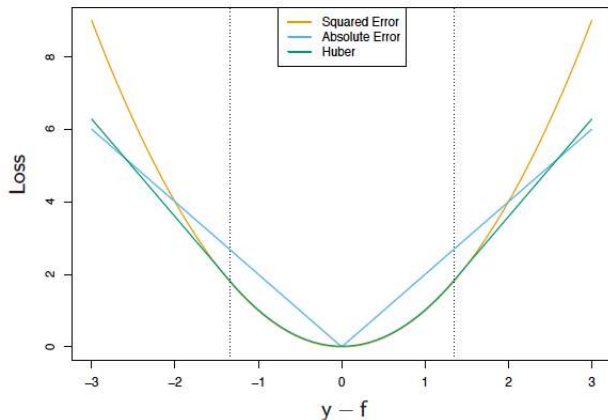


Figure 3: (Hastie et al. 2009:350)

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees**
 - Brief Background on CART
 - Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Boosting Trees

Let's boost some trees!

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees
 - Brief Background on CART
 - Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

What is a Classification/Regression Tree?

- Trees partition the space of all joint predictor variable values into disjoint regions R_j as represented by the terminal nodes of the tree. A constant γ_j is assigned to each such region and the predictive rule is $x \in R_j \Rightarrow f(x) = \gamma_j$

What is a Classification/Regression Tree?

- Trees partition the space of all joint predictor variable values into disjoint regions R_j as represented by the terminal nodes of the tree. A constant γ_j is assigned to each such region and the predictive rule is $x \in R_j \Rightarrow f(x) = \gamma_j$
- So a tree can be formally expressed as:

$$T(x; \Theta) = \sum_{j=1}^J \gamma_j \mathbf{1}(x \in R_j)$$

with $\Theta = \{R_j, \gamma_j\}_1^J$

What is a Classification/Regression Tree

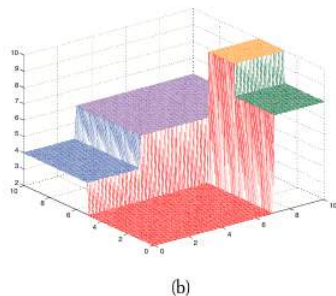
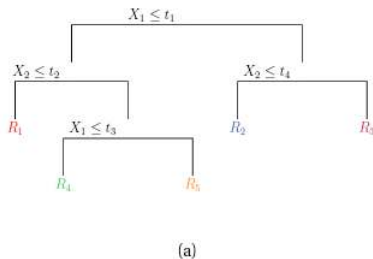


Figure 4: (Murphy 2012:545)

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees**
 - Brief Background on CART
 - Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Boosting Trees

- The boosted tree model is a sum of trees

$$f_M(x) = \sum_{m=1}^M T(x; \Theta_m)$$

Boosting Trees

- The boosted tree model is a sum of trees

$$f_M(x) = \sum_{m=1}^M T(x; \Theta_m)$$

- We will grow our trees in a FSAM. At each step we must solve:

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

Boosting Trees

- The boosted tree model is a sum of trees

$$f_M(x) = \sum_{m=1}^M T(x; \Theta_m)$$

- We will grow our trees in a FSAM. At each step we must solve:

$$\hat{\Theta}_m = \operatorname{argmin}_{\Theta_m} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

- Finding the optimal constants γ_{jm} is easy if we know the regions R_{jm} . But finding the regions is difficult.

Boosting Trees

Two cases where the problem simplifies

- 1 For **squared-error loss** the solution will be to grow a tree that best predicts the current residuals, and $\hat{\gamma}_{jm}$ is the mean of the residuals in each corresponding region.

But recall that neither exponential nor squared-error loss are robust. Choosing other loss criteria, though, make the solution to the problem more difficult.

Boosting Trees

Two cases where the problem simplifies

- 1 For **squared-error loss** the solution will be to grow a tree that best predicts the current residuals, and $\hat{\gamma}_{jm}$ is the mean of the residuals in each corresponding region.
- 2 For **two-class classification with exponential loss** then (under certain conditions) this gives rise to the AdaBoost method for boosting classification trees. In general, the $\hat{\gamma}_{jm}$ will be the weighted log-odds in each corresponding region.

But recall that neither exponential nor squared-error loss are robust. Choosing other loss criteria, though, make the solution to the problem more difficult.

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting**
 - Steepest Descent
 - Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Gradient Boosting

In order to solve this complex problem (for any differentiable loss function) we will implement a solution known as gradient boosting.

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting**
 - Steepest Descent
 - Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Steepest Descent

The solution we will implement is analogous to the steepest descent numerical optimization procedure.

- At any point in the procedure we evaluate the gradient ($\mathbf{g}_m$) of the function $L(\mathbf{f})$ at the last update:

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i)=f_{m-1}(x_i)}$$

Steepest Descent

The solution we will implement is analogous to the steepest descent numerical optimization procedure.

- At any point in the procedure we evaluate the gradient ($\mathbf{g}_m$) of the function $L(\mathbf{f})$ at the last update:

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i)=f_{m-1}(x_i)}$$

- and take a step of length ρ_m , which is the solution to

$$\rho_m = \operatorname{argmin}_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m)$$

Steepest Descent

The solution we will implement is analogous to the steepest descent numerical optimization procedure.

- At any point in the procedure we evaluate the gradient ($\mathbf{g}_m$) of the function $L(\mathbf{f})$ at the last update:

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i)=f_{m-1}(x_i)}$$

- and take a step of length ρ_m , which is the solution to

$$\rho_m = \underset{\rho}{\operatorname{argmin}} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m)$$

- We then update by subtracting $\rho_m \mathbf{g}_m$ from the previous update.

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting**
 - Steepest Descent
 - Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R

Gradient Boosting

- Steepest descent chooses the direction in which the function is most rapidly decreasing

Gradient Boosting

- Steepest descent chooses the direction in which the function is most rapidly decreasing
- We would like to do the same thing but here our solution must be a tree. Also importantly, we don't want to simply minimize loss on the training set but generalize to new data.

Gradient Boosting

- Steepest descent chooses the direction in which the function is most rapidly decreasing
- We would like to do the same thing but here our solution must be a tree. Also importantly, we don't want to simply minimize loss on the training set but generalize to new data.
- A potential solution is to induce a tree at the m th iteration whose predictions $\mathbf{t}_m$ are as close as possible to the negative gradient

$$\tilde{\Theta}_m = \operatorname{argmin}_{\Theta} \sum_{i=1}^N (-g_{im} - T(x_i; \Theta))^2$$

Gradient Boosting

- 1 Initialize $f_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$

Gradient Boosting

- 1 Initialize $f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, \gamma)$
- 2 For $m = 1$ to M :

Gradient Boosting

- ① Initialize $f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, \gamma)$
- ② For $m = 1$ to M :
 - For $i = 1, 2, \dots, N$ compute

$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$$

Gradient Boosting

① Initialize $f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, \gamma)$

② For $m = 1$ to M :

- For $i = 1, 2, \dots, N$ compute

$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$$

- Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \dots, J_m$

Gradient Boosting

① Initialize $f_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$

② For $m = 1$ to M :

- For $i = 1, 2, \dots, N$ compute

$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$$

- Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \dots, J_m$
- For $j = 1, 2, \dots, J_m$ compute

$$\gamma_{jm} = \operatorname{argmin}_{\gamma} \sum_{x_i \in R_{jm}} L(Y - i, f_{m-1}(x_i) + \gamma)$$

Gradient Boosting

① Initialize $f_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$

② For $m = 1$ to M :

- For $i = 1, 2, \dots, N$ compute

$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$$

- Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \dots, J_m$
- For $j = 1, 2, \dots, J_m$ compute

$$\gamma_{jm} = \operatorname{argmin}_{\gamma} \sum_{x_i \in R_{jm}} L(Y - i, f_{m-1}(x_i) + \gamma)$$

- Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x \in R_{jm})$

Gradient Boosting

① Initialize $f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, \gamma)$

② For $m = 1$ to M :

- For $i = 1, 2, \dots, N$ compute

$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$$

- Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \dots, J_m$
- For $j = 1, 2, \dots, J_m$ compute

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(Y - i, f_{m-1}(x_i) + \gamma)$$

- Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x \in R_{jm})$

③ Output $\hat{f}(x) = f_M(x)$

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values**
 - Tree Size
 - Regularization
- 6 Implementation in R

Tuning and Metaparameters

There are a number of parameters for the algorithm that we might be concerned about setting

- J_m , the number of terminal nodes in each tree
- M , the number of boosting iterations
- ν , a shrinkage parameter
- η , fraction of training observations to select at each iteration (stochastic gradient boosting)

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values**
 - Tree Size
 - Regularization
- 6 Implementation in R

Tree Size, J

- Set $J_m = J \forall m$
- Selection of J will affect the number of interactions you allow in your model as the interaction order for any tree is given by $J - 1$.
- HT&F recommend $J \simeq 6$

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values**
 - Tree Size
 - Regularization
- 6 Implementation in R

Regularization: Number of Iterations, M

- In setting the number of iterations we want to run enough to maximally reduce error on the test sample but not so much that we overfit to the training sample.
- This implies some optimal $M = M^*$ that HT&F recommend finding using an early stopping strategy.

Regularization: Shrinkage, ν

- The simplest implementation of shrinkage is to scale the contributions of each tree by a factor $0 < \nu < 1$
- Because smaller values of ν imply a slower learning, there is a tradeoff between M and ν
- HT&F suggest that the best results are found with $\nu < 0.1$

Regularization: Shrinkage, ν

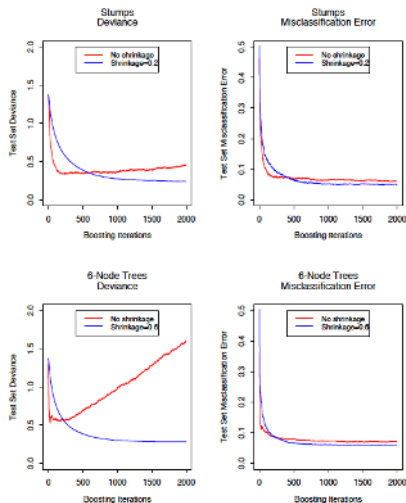


Figure 5: Hastie et al. 2009:366

Regularization: Subsampling, η

- Bootstrap averaging (bagging) can improve the performance of a noisy classifier. We can apply a similar logic here.
- At each iteration we sample without replacement some fraction η of the training observations. A typical value is $\eta = 0.5$. This reduces computational effort while also (often) improving accuracy.

Regularization: Subsampling, η

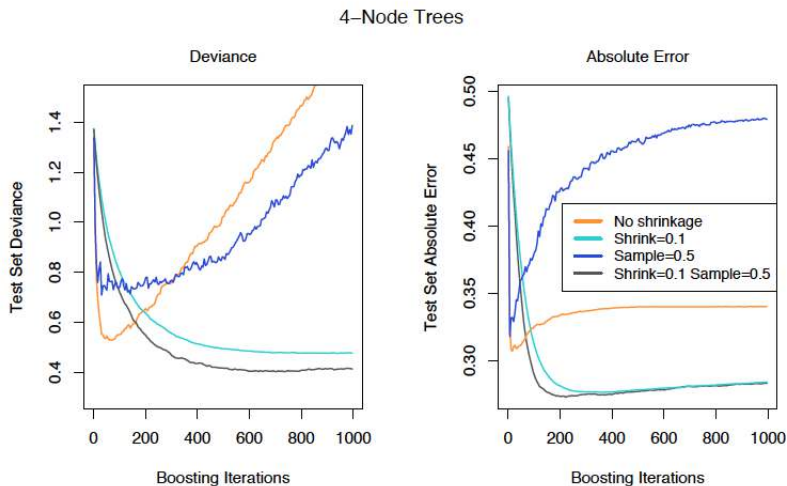


Figure 6: Hastie et al. 2009:367

- 1 AdaBoost
- 2 Selecting a Loss Function
- 3 Boosting Trees
- 4 Gradient Boosting
- 5 Tuning and Metaparameter Values
- 6 Implementation in R**

- `xgboost` implements gradient boosting in R
- command `xgboost` allows you to control the depth of trees, regularization, subsampling, and the number of rounds of boosting
- also creates visualizations of variable importance plots